Composing Web Services using an Agent Factory

Richards, D.J.; van Splunter, S.; Brazier, F.M.; Sabou, M.

published in
Proceedings of AAMAS Workshop on Web Services and Agent-Based Engineering(WSABE), Melbourne, Australia
2003

document version
Publisher's PDF, also known as Version of record

Composing Web Services using an Agent Factory

Debbie Richards
Department of Computing
Macquarie University
Sydney, Australia
richards@ics.mq.edu.au

Sander van Splunter, Frances M.T. Brazier, Marta Sabou
Department of Computer Science
Vrije Universiteit Amsterdam
The Netherlands
<sander, frances, marta> @cs.vu.nl

ABSTRACT
Web service composition can provide a value-chain between customers and suppliers. The increasing number of services, and thus possible combinations, demands the development of dynamic and automatic techniques for their composition. Current commercial solutions are limited and are primarily static and manual. Automation requires reasoning about (semantic descriptions of) the services. In this paper we describe our initial work involving the semantic description of Web services using DAML-S and how our Agent Factory has used these descriptions in its design process to derive a Web service configuration.

Keywords
Web Services, DAML-S, Agent Factory, Semantic Web

1. INTRODUCTION
The web offers organizations the opportunity to reach potential clients like never before through dynamic matching of providers with requestors. However, this promise is also a challenge. The extensive collaborative work that is taking place concerning Web services (WSs) and the Semantic Web (e.g. [1]) is taking up this challenge and is paving the way for realizing dynamic discovery and utilization of web resources. One of the goals of the work reported in this paper is to test the state of the art, identify shortcomings and offer recommendations. Our key goal, however, is to apply and extend our Agent Factory, an automated facility for composing software agents, to use WSs as agent components. Our approach is quite novel in that we treat WS composition as configuration of an artefact. Our findings are of benefit to the web community of users and researchers as we offer a means by which services may be composed into more complex services. In particular, our findings are relevant to the Semantic Web community as we use the semantic descriptions of the web services to determine what components were needed and how they fit together.

In the next section we introduce Web services and the contributions of the Semantic Web community to provide WSs with semantic descriptions. In Section 3 our Agent Factory is introduced and our use of web services as components is described. In Section 4 we offer a scenario for the Agent Factory to solve. We then describe in some detail the steps we took to mark up service descriptions and how these descriptions were used by our Agent Factory to compose a set of WSs. A discussion of the results is given in Section 5. Our conclusions and future work are described in the final section.

2. WEB SERVICE EFFORTS
A number of definitions of Web services exist. The definition that most fits with our intended use of WSs is given by the Stencil[2] group who define WSs as "loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard internet protocols". The definition captures the self-contained, modular, composable and distributed nature of WSs.

The utility of web services relies on the ability to publish, find, browse, select, compose, employ and monitor web services. Current technology relies on humans to perform most of these tasks. For example, while UDDI provides an XML-based schema for describing such aspects as the owner of the service, location and service indexes and categories, to determine if the service is what you need and to actually use it, it is necessary to either make contact with the provider or to browse one or more documents referred to in the UDDI specification. WSDL allows definition of a service interface and service implementation but only at a syntactic level. Similarly, XLANG, WSFL and BPEL4WS only support process modeling at a syntactic level and are too low level (implementation focused) to support reasoning at a conceptual level. Thus having agents automatically find, compose and execute services is not a current reality.

The Semantic Web community has recognized that the key to achieving maximum utility is the addition of semantics to the vast array of resources available via the web[3]. HTML provided "how" to display the data, XML was to offer the "what" description of the data. However, the absence of a shared meaning of the XML tags and namespaces makes dynamic matching between service requestors and service providers impossible. The resource description framework (RDF) and schema (RDFS) offer another layer based on XML that provides some semantics by specifying classes and their properties. DAML+OIL extends the expressiveness of RDF(S) further with several constraints. Currently, what has been learnt from DAML+OIL and industry experience is being combined into the web ontology language (OWL). While these languages are steps in the right direction, a language more specific to the description of web services is needed. DAML-S [9,10] has been developed to provide this and to offer a logically grounded view of web services. The DAML-S service description language includes a number of connected ontologies. We introduce these ontologies and our use of them through our example in Section 4.

DAML-S descriptions have been used by researchers in a number of different projects. Some are actively working on extensions to selected parts of DAML-S. For example, [6] enriches the Process ontology, [22] extends the Profile ontology with bio-informatics related properties and [12] extends the specification of conditions. Some use it for matchmaking with the most notable project being the DAML-S Matchmaker[4]. [18] use an OWL reasoner to perform matchmaking between the DAML-S service descriptions and the user provided criteria. [7] use DAML-S to create an agent wrapper around the service to give it first-order reasoning capabilities. In this way the functionality of the WS is extended

with agent-like behaviour. DAML-S is even being used in OntoMat [11] to support automated extraction of semantics. OntoMat combines the resource with its DAML-S markup. MnM [21] takes automation further and applies techniques from knowledge engineering, machine learning and natural language processing to the DAML-S descriptions to develop rules for marking up other resources.

DAML-S can support composition by providing descriptions that can be used to decide which components may be appropriate, but since *Service* is the highest level concept it is not possible to describe composition of services (composition of processes that make up a service can be described). Therefore, approaches to WS composition using DAML-S must provide a mechanism for their orchestration. One approach is the use of a task language that sits at a layer above DAML-S in the WS architecture. [13] have an agent-based solution that uses DAML(-S) markup of WSs and user constraints. A logic and model-based approach is offered, where a generic model procedure is selected by the user, given to the DAML(-S) enabled agent, who customizes the procedure according to the user specific constraints. The generic procedures are written in an extended version of ConGolog, a situation calculus agent programming language, and executed using a Prolog inference engine. To allow requests for services and dispatch of responses between the web services and agent system, the Open Agent Architecture (OAA) agent brokering system has been used.

Our work is similar to [13] in that we also use generic models to guide the composition. However, by treating WS composition as configuration of a design artefact, the Agent Factory provides an approach that iteratively reasons about requirements, the configuration process and the design artefact and thus takes the whole system development life cycle into account.

3. AGENT FACTORY

An Agent Factory (AF) [4,8] is a servicing facility for automated (re-)design of software agents. Within our Agent Factory [4] a number of main processes are distinguished. For WS composition we are particularly interested in the: (re-)design; assembly; and building block retrieval processes.

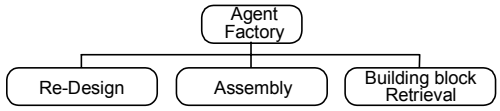

Figure 1. The Agent Factory

(Re-)Design Centre. The (Re-)Design Centre is responsible for the actual (re)design process of agent specifications, based on given requirements, both hard and soft, provided by, e.g., the agent to be redesigned or the owner of the agent.

Building Block Retrieval. Building Block Retrieval is responsible for retrieval of building blocks by querying, in which characteristics with respect to functionality, behaviour and state are specified.

Assembly. In Assembly operational code is assembled on the basis of the operational blueprint produced by the design centre.

This paper focuses only on the design process as fulfilled by the re-design centre. This design process is one of configuration. In the following subsections we consider the (re-)design process, the artefact designed and the use of WS as components.

3.1 The Design Process

The configuration process of a software agent in the (re-)design centre is based on the Generic Design Model (GDM) presented in [5]. In short, the assumption behind this model is that both requirements and their qualifications, and the description of an artefact evolve during a design process. E.g., in practice often not all initial requirements can be satisfied. The artefact is designed to satisfy sets of these requirements. Design choices are influenced by high-level strategies, chosen on process objectives (e.g. deadlines, resources). As shown in Figure 2 this knowledge-based model of design distinguishes reasoning about requirements and their qualifications (Requirement Qualification Set (RQS) Manipulation), reasoning about the design artefact (Design Object Description (DOD) Manipulation), and reasoning about the design process itself (Design Process Co-ordination).

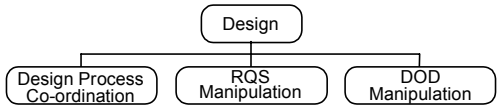

Figure 2. Main processes in GDM

The input and output of all four of these components is defined, together with the level of reasoning (meta-level) to which they pertain. Information exchange between components and potential control structures is also specified by the model as are the necessary control structures and a generic design ontology.

Design in the Agent Factory is conducted at two levels: the conceptual and implementation levels. Adaptation of an agent can involve redesign at both levels and thus requires the mapping between the two levels to be specified explicitly. The operational level includes implementation detail needed by the assembly process.

3.2 The Artefact

One important assumption on which the Agent Factory is based is that the artefacts, the agents, have been designed to be re-designed. This implies, (see [19] for more detail) that (1) agents have a compositional structure with reusable parts; (2) at least two levels of description of agent configurations are defined: conceptual and operational; (3) an ontology to describe the functionality, behaviour and state of agents and their components. The assumption that an agent must have a compositional structure, relies on the availability of compositional models of agents, e.g., ZEUS [14], the GENERIC AGENT MODEL [3], the JADE AGENT MODEL [2], and of compositional models of the tasks agents perform (i.e. knowledge level task models of problem solving methods for generic tasks (IBROW[5], KADS2[6]).

Descriptions of function, state, and behaviour translate directly to the *components*, *data types* and *co-ordination* patterns defined within such models. *Components* refer to the (active) processes distinguished within an agent (which may in turn be composed). *Data types* refer to the information exchanged and manipulated by components. *Co-ordination patterns* are used to define the

temporal sequence and dependencies between tasks (including preconditions and effects). Components, data types and co-ordination patterns are the structures, the building blocks, used within the AF. Ontologies are used to describe them.

The Design Centre of our Agent Factory also uses templates of building blocks. These templates specify both agent models and task models: configurations of the components, data types, co-ordination patterns and ontologies. Templates contain open slots. Slots for building blocks define constraints for the required building block. The slots for components, for example, define the interface, and requirements with respect to the components behaviour and functionality.

3.3 Using Web Services as Components

A key difference between our previous work on the Agent Factory and our current project using web services as components is the reduced level of control and access to the building blocks and the increased number of specification languages in which descriptions may be provided.

The AF operates on building blocks that can denote components, data types or coordination patterns. Web services build on a very similar metaphor: they are components that operate on data-types and can be composed according to some composition patterns. However, because WSs are self-contained, there is a much stronger link between their elements than in the case of the Agent Factory. We treat each web service as a component and consider its inputs and outputs as data types. The internal working of the web-service or a specification of combination of multiple web-services is treated as a coordination pattern. Therefore a web-service is fully described by the combination of the three available types of building blocks.

The next section introduces the DAML-S primitives and describes how they have been used to describe components, data types and coordination patterns at the conceptual and operational levels in the context of our case study.

4. THE STUDY

The assumption that the Agent Factory can be used to configure web services is explored in this section. A specific case study is described for which in section 4.1 a number of web services and their DAML-S descriptions are introduced, and in section 4.2 a trace of the design process for configuring these web services.

4.1 Getting Web Service Descriptions

At the start of this project we looked for web service components that our AF could use. We found that many of the services advertised in UDDI (not just the test sites !!) did not really exist. Some that did exist were unusable when we visited their sites as it was unclear what inputs were expected. If use of a service is difficult with direct human involvement, dynamic composition of services by agents is impossible. We also tried to use services already marked up in DAML-S and found that the few available examples either did not point to real services or did not support a realistic scenario. We therefore decided that we first needed to find a possible suite of services, mark them up and see whether our agent was able to use the mark up. These services can be used individually or composed in a variety of ways to develop a browseable web portal of references in BibTEX as shown in Figure 3. Combinations depend on whether the initial file is already in RDF format, there are multiple files that need to be added to the SESAME repository, the data is already in the SESAME repository, and so on.

The five services[7] that are used in the scenario are:

- Bib2Rdf (B2R) – a file conversion service that takes a file in BibTEX format and outputs a file in RDF format
- ISESAME – a file import service that takes a file in RDF format and adds it to a specified public or private repository in SESAME[8].
- ESESAME – a file export service that extracts data from a specified public or private repository in SESAME and outputs the data in RDF, n3 or ntriples.
- SameIndividualAs (SIA) – a utility service that reads in a file in RDF format and adds the *sameIndividualAs* DAML tag to duplicate names.
- PortalCreator - AIdMinistrator[9] (AI-DIS) – a service which takes the contents of a SESAME repository and displays it in a web portal.

Currently to use these services it is necessary to manually invoke them individually in the desired order or to run a script that will invoke the services in a prescribed order. Our goal was to automatically configure these services using the Agent Factory according to the particular situation and user's requirements.

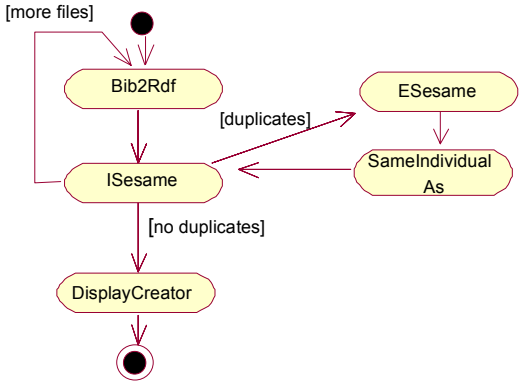

Figure 3. The Process of Developing a Display Portal using AIdMinistrator

In this section we describe the various models we needed to create in order to reason about the services in the AF.

4.1.1 Adding Semantics to Web Services Descriptions

To describe our web-services we used elements from the four ontologies that constitute DAML-S: *Service(ser:), Profile(p:), Process(pr:) and Grounding(gr:)*. Additionally, we specify domain knowledge in a fifth, *Domain(d:)* ontology. In this paper, the text in brackets prefixes elements declared in the corresponding ontology. Each of these ontologies is described below with an example of how we have used them for our simplest service: B2R.

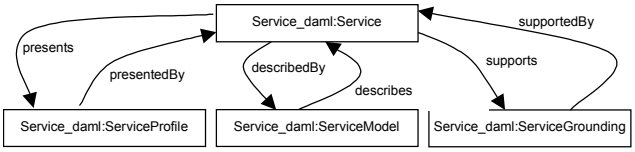

Figure 4. DAML-S ontology

4.1.1.1 Service Ontology – the WS parts

This ontology specifies the highest level DAML-S concept: a Service. Further it specifies the relation of the Service class to other classes (see Fig. 4). Each of these describes the service from a certain point of view using concepts defined in the corresponding DAML-S ontology. A ServiceProfile describes the capabilities of the service for discovery purposes while a ServiceModel details its internal workings. A ServiceGrounding links the abstract description of the service to actual implementation details, such as message exchange formats and network protocols so that automatic invocation of the service is possible. Currently support is offered for grounding DAML-S to WSDL service descriptions.

The B2R service is modeled as a Service instance (see specification below) that is connected to three instances describing the ServiceProfile (Profile_B2R), ServiceModel (Process_B2R) and ServiceGrounding(Grounding_B2R).

```
<ser:Service rdf:ID="Service_B2R">
 <ser:presents rdf:about="#Profile_B2R"/>
 <ser:describedBy rdf:about="#Process_B2R"/>
 <ser:supports rdf:about="#Grounding_B2R"/>
</ser:Service>
```

4.1.1.2 Profile Ontology – what the WS does

The Profile ontology contains the vocabulary to describe the ServiceProfile. Its central concept, Profile, is a subclass of ServiceProfile and contains the contact information of providers, an extensible set of service characteristics and a functionality description by specifying the inputs, outputs, preconditions and effects of the service (IOPE's). It also points to the described Process (see 4.1.1.3).

We primarily used the parts of the functional description and took special care to link them to domain concepts. First, the Profile_B2R instance is of type "Translator" which is a domain specific `p:Profile` (see the Domain Ontology). Second, we used domain concepts for the IOPE's. The DAML-S Profile ontology models the IOPE's as subproperties of `p:parameter`. Each parameter has a `ParameterDescription` which mandates describing the parameter through the `p:restrictedTo` property (see DAML-S specification bellow). The value of this property is a domain concept that gives the "semantics" of the parameter. For our service the input is encoded in BibTex, contains references and is provided as a URL. In our domain ontology we collectively refer to such information sources as `RefBibURL`. Similarly the output is `RefRDFStream`, i.e. an RDF Stream of references.

```
<d:Translator rdf:ID="Profile_B2R">

<!-- reference to the service -->
<ser:presentedBy rdf:resource="#Service_B2R "/>

<!-- reference to the process -->
<p:has_process rdf:resource="#P1"/>

<!-- Descriptions of IOPEs -->
<p:input>
<p:ParameterDescription  rdf:ID="BibTex_File">
 <p:restrictedTo rdf:resource="&d;#RefBibURL"/>
 <p:refersTo rdf:resource="#proc_bibFile_In"/>
</p:ParameterDescription>
</p:input>

<p:output>
<p:ParameterDescription  rdf:ID="Rdf_File">
 <p:restrictedTo rdf:resource="&d;#RefRDFStream"/>
 <p:refersTo rdf:resource="#proc_rdfFile_Out"/>
</p:ParameterDescription>
</p:output>

</d:Translator>
```

Other parts of this description express the link between the Profile and Process Models. We will describe them after presenting the Process Ontology.

4.1.1.3 Process Ontology – how the WS works

A ServiceModel is further specialized as a ProcessModel in the Process ontology. A ProcessModel has a single Process which can be atomic, simple or composite (composed from atomic processes through various control constructs such as split, unordered, iterate, etc). The `Process_B2R` ProcessModel has a single atomic process, P1.

Each Process has a set of IOPE's. These are modeled as properties of the Process and take as value a domain concept that describes them. For example, `proc_bibFile_In` is an input of the process P1 described by the `RefBibURL` domain concept.

```
<pr:ProcessModel rdf:ID="Process_B2R">
  <pr:hasProcess rdf:resource="#P1" />
  <s:describes rdf:resource="#Service_B2R "/>
</pr:ProcessModel>

<!-- Definition of top level Process-->
<daml:Class rdf:ID="P1">
<daml:subClassOf rdf:resource="&pr;#AtomicProcess"
/>
</daml:Class>

  <!-- IOs are properties for the process-->
<daml:Property rdf:ID="proc_bibFile_In">
 <daml:subPropertyOf rdf:resource="&pr;#input"/>
 <daml:domain rdf:resource="#P1"/>
 <daml:range rdf:resource="&d;#RefBibURL"/>
</daml:Property>

<daml:Property rdf:ID="proc_rdfFile_Out">
 <daml:subPropertyOf rdf:resource="&pr;#output"/>
 <daml:domain rdf:resource="#P1"/>
 <daml:range rdf:resource="&d;#RefRDFStream "/>
</daml:Property>
```

The ServiceProfile and ServiceModel are two different descriptions of the same service, and naturally links exist between them. Identifying these links and specifying them correctly ensures the consistency of the description. Firstly, each Profile instance states the Process it describes through the property `has_process`. In our case `Profile_B2R` describes the P1 process. Secondly, the IOPE's of a Profile correspond to the IOPE's of the described Process. We use the `p:refersTo` property to specify the Process parameter corresponding to the given profile parameter.

4.1.1.4 Grounding Document – how to use the WS

The Grounding ontology specializes the ServiceGrounding as WSDLGrounding containing a set of WsdlAtomicProcess-Grounding elements. In our case the service grounding `Grounding_B2R` consists of a single AtomicProcessGrounding.

```
<gr:WsdlGrounding rdf:ID="Grounding_B2R">
 <s:supportedBy rdf:resource="#Service_B2R"/>
  <!--Collecton of all groundings -->
  <gr:hasAtomicProcessGrounding rdf:res="#Gr1"/>
</gr:WsdlGrounding>
```

+p:Profile
 +InfoService
  +Translator
  +RedundancyChecker
  +Repository
   +SesameStorer
   +SesameExtractor
  +DisplayCreator
   +AidmDisplayCreator
+InfoSource
 +RefRdfFile
 +RefBibURL
 +RefRdfStream
 +RefIS
 +SiaRdfStream
 +RdfStreamIS
 +FbRdfStream
 +QryIS
 +QryRdfStream
 +URLIS
+Content
 +References
 + Query
 + Feedback
 + SIA
+Syntax
 +RDF
 +BibTex
+Type
 +Stream
 +URL
 +File
Display
 +Portal

Figure 5: The domain ontology (concept hierarchy)

We do not present more of the Grounding specification given its complex syntax. To ground a DAML-S specification to WSDL, each atomic process defined in the Process model is grounded to a WSDL operation. Also, each input (output) of the process becomes a message part in the input (output) message of the corresponding operation. The WSDL file specifies the type of its message parts in terms of xml dataTypes. Further, in the WSDL file the transport protocol, the messaging format, and the actual network addresses of the service are given.

4.1.1.5 Our Domain Ontology

To allow matching at the semantic level it is important to define the terms that are appropriate for the given domain. In our domain ontology we define concepts that best describe our services. We provide concepts for different types of Information Services (InfoService) and for defining an Information Source (InfoSource). For our domain, the following characteristics of each InfoSource are important: its content (describing the contained data), its syntax (RDF/BibTex) and its type (File/URL/Stream). We model this with three properties: hasContent (ranges over Content instances), hasSyntax (ranges over Syntax instances) and hasType(ranges over Type). All three have InfoSource as their domain.

With this construction we can define special InfoSources by imposing range restrictions on the three properties. For example an RDF File containing references (RefRdfFile) is a subclass of InfoSource such that the ranges of the three main properties are restricted accordingly. We provide the DAML specification for such a class.

```
<daml:Class rdf:ID="RefRdfFile">
 <rdfs:subClassOf  rdf:resource="#InfoSource"/>

 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#hasContent"/>
   <daml:toClass rdf:resource="#Reference"/>
  </daml:Restriction>
 </rdfs:subClassOf>

<rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#hasSyntax"/>
   <daml:toClass rdf:resource="#RDF"/>
  </daml:Restriction>
 </rdfs:subClassOf>

<rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#hasType"/>
   <daml:toClass rdf:resource="#File"/>
  </daml:Restriction>
 </rdfs:subClassOf>

</daml:Class>
```

We use the concepts from the domain ontology to augment the descriptions of the Profile and Process model. As such they provide the basis for reasoning in the Agent Factory configuration process described in the next section.

Table 1: Web services within the repository

	Conceptual Building Block – from Profile				Operational BB – from Grounding	
Service	task	Input	Output	Pre conditions	Input	Output
AI-DIS	AIdmDisplay Creator	RefIS	Display	- info must be in SESAME repository - input must have unique identifiers for authors	retrieves own input (see operational precondition)	none
SIA	Redundancy Checker	RefRdfStream	SiaRdfStream	whole set of references must be presented	xsd:string	xsd:string
ISESAME	SESAMEStore r	RdfStreamIS	FbRDFStream	SESAME handles daml-tag:sameIndividualAs-tag if this is added beforehand.	xsd:string	xsd:string
ESESAME	SESAME Extractor	QryRDF Stream	RdfStreamIS	SESAME handles daml-tag if this is added beforehand.	xsd:string	xsd:string
B2R	Translator	RefBibURL	RefRdfStream		xsd:URL	xml:string

Table 2: Templates within the repository.

Template	Slots	Input of component	Output of component
display data_1	retrieve data	URLIS	IS
	store data	IS	IS
	create display	IS	Display
display data_2	retrieve data	IS	IS
	store data	IS	IS
	filter data	IS	IS
	create display	IS	Display
handle data storage	store data	IS	none required
	extract data	QryIS	IS

4.2 Using Service Descriptions for Configuration

This section illustrates the use of the Agent Factory for web service configuration. A trace of the design process as used in the AF for the configuration of the web services described in the previous section is used for this purpose.

4.2.1 Introduction of trace

The web-services of this scenario are summarised in Table 1. Web service descriptions include a conceptual description and an operational description. The mapping between conceptual and operational descriptions (normally an issue within the AF) is therefore trivial: it is not only one-to-one and structure preserving, it is pre-defined. The templates used in this scenario are summarised in Table 2. The names of the slots are stating the required functionality of the building blocks to be inserted. Furthermore the template has constraints on the conceptual data types that are exchanged by its slots. In this example the templates do not offer constraints for operational data types.

For conceptual data types used in the trace IS subtypes are defined in the domain ontology of Figure 5. As an example, the data type RefIS, is an InfoSource which only restricts the Content to references, and makes no constraints on Syntax and Type of InfoSource. QryIS refers to an InfoSource containing a Query, and FbRdfStream refers to an InfoSource with the Content Feedback, the Syntax RDF, and the Type Stream.

As briefly described in Section 3.1, the Agent Factory uses the Generic Design Model as the basis for the design process. This model explicitly distinguishes three processes: reasoning about the design process (in DPC), reasoning about requirements and their qualifications (in RQSM), and reasoning about the design object description (in DODM). These processes and the interaction between these processes are distinguished in the trace.

4.2.2 The trace

The initial problem stated by a user is:

> "Display bibliographical references available in BibTeX, per author using AIdMinistrator services".

This translates to the following formal requirements (RQS_init):

rq_i1 display publications per author
rq_i2 create display from BibTexfiles
rq_i3 use AIdMinistrator services for display

The design process starts from scratch, implying that no initial design object description (DOD_init) is available.

Given the initial input the first process to be activated is DPC. DPC, in turn, activates RQSM . Given
RQS_init = {rq_i1, rq_i2, rq_i3}
RQSM concludes
process_eval0 = set of requirements received
DPC then activates DODM, that given
DOD_init = empty
concludes:
DOD_process_eval0 = no initial DOD available

The design process can now really start. DPC activates RQSM to determine model-level requirements:
RQSM_strat1 = focus on model-level requirements

RQSM discovers that the requirements provided are too specific. It uses the knowledge it has of model-level requirements to generalize requirement *rq_i2* to
rq4 = display data
and determines that this model level requirement is its output:
RQS_1= {rq4}
Note that RQS_1 does not contain the initial requirements. RQSM has determined the set RQS_1 as model-level requirements, so the strategy has been fulfilled successfully, creating the evaluation output:
RQSM_process_eval1 = RQSM_strat1 succeeded

On the basis of this output DPC observes that RQSM has successfully finished, and DPC activates DODM to initially focus on the conceptual design:
DODM_strat1 = focus on conceptual design

DODM receives its input RQS_1 from RQSM and its strategy DODM_strat1 from DPC. DODM initial attempt to find the simplest template that fulfils the requirement specified by RQS_1, namely to *display data* is successful. This template is depicted in Figure 6.
The template display data_1 has three open slots for components: retrieve data, store data, and create display. Each slot also specifies an interface (input and output). The retrieve data slot expects URLIS as input and provides InfoSource (IS) as output. The store data slot expects IS as input and provides IS as output. The create display slot expects IS as input and produces a display as output.

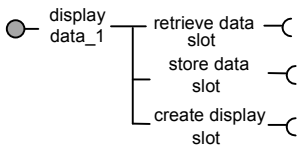

Figure 6: Conceptual template for data display

As DODM has successfully found a model for the conceptual design fulfilling RQS_1
RQS_eval1 = RQS_1 completely satisfied by DODM
DODM_process_eval1 = DODM_strat1 succeeded

DPC proceeds to activate RQSM to determine which other requirements need to be considered.

From hereon a summarized, more abstract description of the trace of the configuration process for templates and components is depicted.

RQSM identifies rq_i3 as a very specific requirement for the display creation. This requirement (*use AIdMinstrator services for display*) is input for DODM, which is activated by DODM now that RQSM has finished. It limits the search space considerably. Only one single web-service fulfils the given requirements. This service has an operational and a conceptual description. The conceptual part of AI-DIS is inserted in the *create display slot*. The data-exchange is checked for this slot: RefIS is a sub-type of IS, and portal is a sub-type of a display, therefore the conceptual data-exchange for this slot is correct. Apparently the goal is not to display data, but, more specifically, to display references. In the template the outputs of the *store data slot*, and in- and output of the *retrieve data slot* are refined from IS to RefIS (not shown in a figure).

The template is not yet complete and DPC activates RQSM. By adding components to the design also new constraints in the form of pre-conditions come into play. The precondition "info must be in SESAME repository" of AI-DIS is relevant for selecting a component for *store data slot.* A SESAME repository is used for storage. RQSM transforms the pre-condition into a requirement "store data in SESAME repository" and passes this requirement to DODM. DODM is activated by DPC. The requirement provided by RQSM focuses the adaptation to *store data slot*. The references need to be stored in SESAME, before the display is created. A separate service for data extraction from SESAME is not required for AI-DIS. The web-service ISESAME is selected and used for the *store data slot* within the template, as shown in Figure 7. The conceptual input and output are both specialized from RdfStreamIS to RefRdfStream, for information on references will be stored. This addition has also implications for *retrieve data slot* for which the component will have to produce a RefRdfStream.

Figure 7: Conceptual partially specified web-service composition

The template is still not complete and DPC activates RQSM. RQSM deduces that rq_i2 is now relevant, for it expresses the format of the input (BibTex files) for the component that is to be inserted in the *retrieve data slot*, rq_i2 is passed to DODM. DODM can now refine the input for the component for the *retrieve data slot* from URLIS to RefBibURL. The component to be sought to retrieve data needs to produce RefRdfStream from instances of RefBibURL. The configuration is extended by the service B2R which matches the requested specification. Its conceptual description is inserted in the *retrieve data slot*, as shown in Figure 8. DODM finishes reasoning.

Figure 8: A conceptual completed template for displaying data

The template has no open slots left. DPC decides to now focus on constraints imposed by the co-ordination patterns (i.e. the pre- and post-conditions specified for each slot in a template). RQSM transforms all the remaining pre-conditions to hard requirements (as done in the previous step with the pre-condition): The references presented to AI-DIS must have unique identifiers for each author, and SESAME can handle the DAML:sameIndividualAs-tag when this tag is added to equal authors beforehand. DODM receives these requirements. The problem for the input is AI-DIS is solved by ISESAME which can handle the *DAML:sameIndividualAs*-tag, but this tag needs to have been added beforehand to equal authors. The addition of the service SIA would take care of tagging these instances. However, there are no open slots left in the current template. This implies that the current artefact does not suffice and a new design process is initiated given the current design and the conflicting requirements. The current template for display creation is replaced by a more complex template for display creation that also enables the manipulation of data, shown in Figure 9.

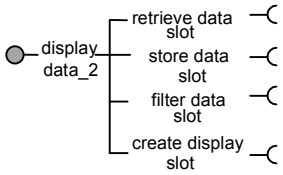

Figure 9: Template for display creation, enabling manipulation of data

Most of the current design, as shown in Figure 8 can be reused, however the precondition of the web-service SIA is that the whole collection of references must be presented as input for this. Whereas the design in Figure 8 enables references to be stored, it does not enable them to be extracted. Therefore the *store data slot* needs be filled with a template that can both store and extract data. The conceptual template for this task is shown in Figure 10.

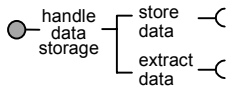

Figure 10: Template for data storage handling

The slots in the template handle data storage are filled with the components ISESAME and ESESAME. To complete the final configuration the template from Figure 9 is filled reusing the design of Figure 8, and inserting SIA in the *filter data slot*. The result of the final configuration shown in Figure 11.

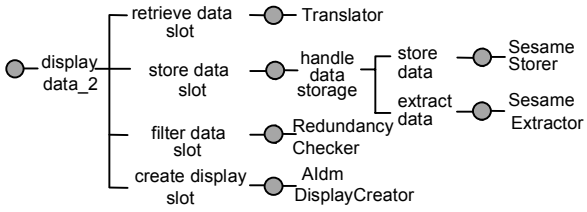

Figure 11: Final configuration of conceptual building blocks

Due to the nature of the mapping between conceptual and operation descriptions (one-to-one and thus structure preserving as discussed above) slots in the conceptual templates are also slots in the operational templates. The operational descriptions of the WS have been simplified in this example: almost all operational inputs and outputs have been specified as simple xsd:strings. In more complex circumstances, the operational level often can reveal syntactical differences. For example two conceptually identical concepts may be mapped to different syntactical entities. This requires more extensive reasoning about the operational level. In our simplified description of the trace all requirements are now, however, satisfied. DPC terminates the design process.

The trace has successfully concluded, creating a specification for a configuration of web-services, and using templates to achieve this configuration.

5. DISCUSSION

In the previous section we have shown how the AF can reason about the design process, requirements and design artefact to produce a configuration of web services. While the knowledge used in the configuration process will change according to the domain, and typically be specified in a domain ontology, we found that the AF architecture and design process did not need to change. The re-design process creates descriptions of the composition, and the assembly is trivial, when the produced description is directly interpretable, as is the case with the DAML-S descriptions.

We have used templates for the configuration. The partial specification has been completed using WSs. In this simple scenario, the DAML-S descriptions provided sufficient information for reasoning purposes. More specifically, the Profile models provided descriptions of the components at the conceptual level and the Grounding model described the service at the implementation level. Data types were also specified at both levels in these models through the definition of IOs and various external ontologies (such as xsd, our domain ontology, rdfs and the DAML-S upper level ontologies themselves). Coordination patterns were supported via the ControlConstructs for composite processes in the Process ontology and the definition of preconditions and effects. PE's can be used to express dependencies between web-services. Within the Profile description, only PE's concerning the usage of the whole service are specified. The Process description can describe specific PE's that occur during the use of the web-service. Note that the Process description of a composite process using ControlConstructs and PEs, is a single coordination pattern that has been instantiated and is only useful possibly for matching or monitoring purposes but not for dynamic composition.

We offered an alternative solution to the Publication Portal Creation problem given in Section 4 in [20]. In that paper, we provided a solution in the context of resources constrained to components/services described in DAML-S, that is, templates were not available. Without templates the process was essentially one of constraint satisfaction on IOs. We are currently implementing our solution and developing additional WSs and DAML-S descriptions to support a wider range of scenarios. We expect some revisions to our solution to deal with implementation issues.

Through exploration of the literature and hands-on experience we have some comments regarding the UDDI, WSDL and DAML-S matchmaker. Finding services in UDDI is indeed difficult. tModels as a solution to providing semantics is limited to pointing to another resource that might contain semantics. [15] have sought to extend the usefulness of the UDDI tModel by introducing the idea of tModels for each DAML-S parameter. This is a possible but not very elegant solution. The handling of semantics within UDDI needs considerable rethought. The link between WSDL and UDDI is well defined and documented. However, the links between the various models can be quite confusing and disastrous if incorrect. Tools are offered which can minimize errors. The specification of SOAP messages in WSDL is well documented but less assistance is provided for HTTP messages, which meant more effort on our part. The DAML-S Matchmaker offers a good service and an easy to use form-filling interface. However, we found finding services difficult and still required a level of knowledge of the service being sought. In particular, knowledge of the relevant ontologies used to describe the service was needed to enable matchmaking to succeed. We would also like to see matchmaking extended to the Process model. This would be used as a second step after matching on the Profile to assist selection from a set of alternatives. For example, the order in which atomic/composite processes occur may affect the suitability of the service for a particular problem.

Regarding DAML-S we have numerous observations, given in more detail in [16]. The use of DAML-S is not always straightforward. The conceptual model underlying DAML-S is imprecise. Different parts of the language build on different metaphors (action/ function). The links between these conceptual models are poorly specified and often inconsistent. Within DAML-S there is also little reference to standard Software Engineering terminology: while basic concepts are employed (such as "function/ method"), there are no directions given about how to model more complex situations (such as parametric polymorphism). This imprecise conceptual model provided flexibility in modelling, but more often it led to confusion.

Given that the Process model only expresses a single coordination pattern, we are reviewing the role and value of the DAML-S Process ontology. If the Process model is only useful for monitoring a single web service, perhaps the overhead of developing a Process model is not warranted. We are considering whether the four DAML-S ontologies could be replaced via extending WSDL with conceptual level semantic properties to produce a description of the service far less verbose than the DAML-S ontology.

Some further open issues for us concerning the use of DAML-S within the AF are:

- how to handle the problem of multiple interfaces to the same service without having to create a separate service for each interface ?
- how to define services in terms of other (complementary, supplementary, related, etc) services which can be useful in selection and composition of services in the agent design ? and
- are the properties that can be defined/searched in the DAML-S specification adequate for service discovery and composition of more complex tasks?

The partial solution to the first problem offered by [1] is to define the IOPEs in the Service profile at a higher level of abstraction that would cover the range of possible IO types. For matching purposes an overly general concept may not be useful and multiple specializations will not be possible as multiple inheritance is not allowed. The solution offered by [1] also does not cater for the situation where parameters are different in order or number.

6. CONCLUSIONS AND FUTURE WORK

In summary the DAML-S WS description language is sufficiently expressive for specifying conceptual and operational building blocks for the simple services in our example. However, we would like to see review and possible extensions to DAML-S to support description of:

- multiple interfaces without having to specify multiple profiles, processes and groundings;
- collections of services or at least allow relationships between services to be described;
- capabilities beyond IOPEs and type of service.

Existing commercial approaches tend to be process-flow oriented and do not support dynamic and automatic WS composition. We have proposed a solution based on the Agent Factory which treats composition as adaptation of a template or design pattern by reasoning about requirements, the process and the design artefact. In this paper, we have considered the use of WSs to be the components that complete the design. This leaves the question of where to find templates. A number of researchers have conducted work that we will explore further, for example:

- the earlier work on problem solving methods [17] and generic task models [3];
- The work of [8], also known as the Agent Factory, which includes the PASSI methodology and an extended-UML CASE tool to help human designers design an agent. The various diagrams may be compiled to generate an agent skeleton, database of patterns, reports and design documents;
- The IBROW[5] project which semi-automatically configures intelligent problem solvers using problem solving methods as building blocks.

In addition to templates which specify agent models and task models, for the purpose of WS composition, our AF should include WS models. A possible source of such templates could be to mine them from Process models.

7. ACKNOWLEDGMENTS

Our thanks to DAML-S coalition and the SW@VU for providing the bases for our explorations, and to N.J.E. Wijngaards for his work on the Agent Factory.

8. REFERENCES

[1] Ankolekar, A., Huch, F. and Sycara, K. Concurrent Execution Semantics for DAML-S with Subtypes, *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.

[2] Bellifemine, F., Poggi, A., and Rimassa, G., Developing multi-agent systems with a FIPA-compliant agent framework. *Software - Practice and Experience* 31(2): 103-128, 2001.

[3] Brazier, F.M.T., Jonker, C.M., Treur, J.: Principles of Component-Based Design of Intelligent Agents. Data and Knowledge Engineering 41 (2002) 1-28.

[4] Brazier, F.M.T., Wijngaards, N.J.E. Automated Servicing of Agents *AISB Journal, Special Issue on Agent Technology*, 1:1 (2001) 5-20.

[5] Brazier, F.M.T, Van Langen, P.H.G., Ruttkay, Zs. and Treur, J. On formal specification of design tasks *In Proc. of the AAAI Workshop on Artificial Intelligence and Manufacturing: State of the Art and Practice*, AAAI Press, 1994, 30-39.

[6] Brison, J., Martin, D., McIlraith, S. and Stein, L.A., Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web. http://www.cs.bath.ac.uk/ jjb/ftp/springer-daml.pdf.

[7] Buhler, P. A. and Vidal, J. M. (b) Semantic Web Services as Agent Behaviors. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, editors, *Agentcities: Challenges in Open Agent Environments*, pages 25-31. Springer-Verlag, 2003.

[8] Cossentino, M. Burrafato, P., Lombardo, S. and Sabatucci, L. Introducing Pattern Reuse in the Design of Multi-Agent Systems. *AITA'02 workshop at NODe02* - 8-9 October 2002 - Erfurt, Germany.

[9] DAML Services Coalition. DAML-S: Semantic Markup for Web Services. DAML-S v. 0.7 White Paper, October 2002.

[10] DAML-S Coalition. Describing Web Services using DAML-S and WSDL. DAML-S Coalition working document, http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html, August 2002.

[11] Handschuh, S., Staab, S. and Maedche, A. CREAM-Creating Relational Metadata with a Component-Based, Ontology-Driven Annotation Framework, *1st Int. Conf. on Knowledge Capture (K-CAP'2001), Workshop on Semantic Markup and Annotation*, Victoria, BC, Canada, October 2001.

[12] Lopes, A., Gaio, S. and Botelho, L.M.,. From DAML-S to Executable Code. In *Proc. of the Workshop Challenges in Open Agent Systems AAMAS 2002*.

[13] McIlraith, S., Son, T.C. and H. Zeng, H., Mobilizing the Semantic Web with DAML-Enabled Web Services, *Proc.*

Second Int'l Workshop Semantic Web (SemWeb'2001), Hongkong, China, May, 2001.

[14] Nwana, H.S., Ndumu, D.T., Lee, L.C. ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems. *Applied AI* 13:1/2, 1998, 129-185.

[15] Paolucci M., Payne T., Sycara K. and Zeng H. 2001. DAML-S: Semantic markup for Web services. In *Proc. of the International Semantic Web Working Symposium*, Stanford, CA.

[16] Sabou, M., Richards, D. and van Splunter, S. An experience report on using DAML-S, Submitted to *Workshop on E-Services and the Semantic Web*, Budapest, Hungary, May 2003.

[17] Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W., Wielinga, B.: *Knowledge Engineering and Management, the CommonKADS Methodology*. MIT Press, 2000.

[18] Sirin, E., Hendler, J. and Parsia, B. Semi-automatic Composition of Web Services using Semantic Descriptions. Accepted to *Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, 2002.

[19] Splunter, S. van, Wijngaards, N.J.E., Brazier, F.M.T., Structuring Agents for Adaptation In Alonso, E., Kudenko, D., Kazakov, D. (eds.) *Adaptive Agents and Multi-Agent Systems*, Lecture Notes in Artificial Intelligence (LNAI) 2636, Springer-Verlag Berlin. 2003.

[20] Splunter, S. van, Sabou, M., F.M.T. Brazier and Richards, D. Configuring Web Services, using Structuring and Techniques from Agent Configuration, *EEE/WIC International Conference on Intelligent Agent Technology (IAT 2003)* (submitted).

[21] Vargas-Vera, M, Motta, E., Domingue, J, Lanzoni, M., Stutt, A. and Ciravegna, F. MnM: Ontology Driven Tool for Semantic Markup. *European Conference on Artificial Intelligence (ECAI 2002). In proceedings of the Workshop Semantic Authoring, Annotation & Knowledge Markup (SAAKM 2002)*. Lyon France, July 22-23, 2002.

[22] Wroe, C., Stevens, R.,Goble, C., Roberts, A. and Greenwood, M., A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data *Journal of Cooperative Information Science*, 2003.

---

1 http://www.semanticweb.org
2 http://www.stencilgroup.com/ideas_scope_200106wsdefined.html
3 http://www.daml.org/services/
4 http://www-2.cs.cmu.edu/~softagents/daml_Mmaker/daml-s_matchmaker.htm
5 http://www.swi.psy.uva.nl/projects/IBROW3/home.html
6 http://www.swi.psy.uva.nl/projects/CommonKADS/home.html
7 these services are part of the SW@VU project http://www.cs.vu.nl/~mcaklein/SW@VU/
8 http://sesame.aidministrator.nl
9 the component used for the display/portal creation is developed by AIdMinistrator http://www.aidministrator.nl